Machine Learning HW 4 Report

0516215 林亮穎

1. Forward-propagate code

```python
def forward_propagate(X, theta1, theta2):
    m = X.shape[0]
    X = np.matrix(X)

    #Write codes here
    a1 = np.concatenate((np.ones((m, 1), dtype=float), X), axis=1)              # 5000x401
    z2 = np.dot(a1, theta1.T)                                                   # 5000x401 * 401*10 = 5000*10
    a2 = np.concatenate((np.ones((m, 1), dtype=float), sigmoid(z2)), axis=1)    # 5000x11
    z3 = np.dot(a2, theta2.T)                                                   # 5000x11  * 11x10  = 5000x10
    h  = sigmoid(z3)                                                            # 5000x10

    return a1, z2, a2, z3, h
```

2. Back-propagate code

```python
def backprop(params, input_size, hidden_size, num_labels, X, y, learning_rate):
    m = X.shape[0]  #5000 testcases

    #Write codes here

    theta1 = np.matrix(np.reshape(params[:hidden_size * (input_size + 1)], (hidden_size, (input_size + 1))))
    theta2 = np.matrix(np.reshape(params[hidden_size * (input_size + 1):], (num_labels, (hidden_size + 1))))
    nabla1 = np.zeros((hidden_size, input_size+1))  # 10x401
    nabla2 = np.zeros((num_labels, hidden_size+1))  # 10x11

    for i in range(0, m):

        # STEP1: Forward Propagation
        # a1, z2, a2, z3, h = forward_propagate(X, theta1, theta2)
        tmp = X[i,:]
        tmp.shape
        a1 = np.c_[np.ones((1,)), X[i, :]]                  # 1x401
        z2 = np.dot(a1, theta1.T)                           # 1x401 * 401x10 = 1x10
        a2 = np.c_[np.ones((1,)), sigmoid(z2)]              # 1x11
        z3 = np.dot(a2, theta2.T)                           # 1x11  *  11x10 = 1x10
        h  = sigmoid(z3)                                    # 1x10

        # STEP2: Calculate delta3
        delta3 = h - y[i, :]                        # 1x10
        delta3 = delta3.T                           # 10x1
        delta3.shape

        #STEP3: Calculate delta2
        # delta2 = np.dot((theta2[:,1:]).T, delta3) * sigmoid_gradient(z2).T   # 10x10 * 10x1 = 10x1
        delta2 = np.multiply(np.dot((theta2[:,1:]).T, delta3) , sigmoid_gradient(z2).T)
        # delta2 = np.dot((theta2[:,1:]).T, delta3)   # 10x10 * 10x1 = 10x1


        #STEP4: Accumulate the gradient
        nabla1 = nabla1 + np.dot(delta2, a1)   # 10x1 * 1x401 = 10x401
        nabla2 = nabla2 + np.dot(delta3, a2)   # 10x1 * 1x11  = 10x11
```

```
130
131     # STEP5: Obtain the gradient
132     grad1 = (1.0/m) * nabla1   # 10x401
133     grad2 = (1.0/m) * nabla2   # 10x11
134
135     lambda_ = 1
136     grad1[:, 1:] = grad1[:, 1:] + (lambda_/m) * theta1[:, 1:]  # do not regularize bias
137     grad2[:, 1:] = grad2[:, 1:] + (lambda_/m) * theta2[:, 1:]
138
139
140     grad = np.hstack((grad1.ravel(), grad2.ravel()))
141
142     params_trained = np.hstack((theta1.flatten(), theta2.flatten()))
143     J = cost(params_trained, input_size, hidden_size, num_labels, X, y, learning_rate)
144
145     return J, grad
```

3. Result accuracy

```
accuracy = 97.34%
```